МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Програмний комплекс симуляції бойової обстановки. Захищена серверна частина»

Завідувач випускаючої кафедри — **Довбиш А.С.**

Керівник роботи — **Лаврик Т.В.**

Студента групи ІН – 61 — **Шелест С.М.**

СУМИ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2020 р.

ЗАВДАННЯ

до випускної роботи

Студента четвертого курсу, групи КБ-61 спеціальності "Кібербезпека" денної форми навчання Шелеста Сергія Миколайовича

Тема: "Програмний комплекс симуляції бойової обстановки. Захищена серверна частина"

Затверджена наказом по СумДУ

№_____ від _____2020 р.

Зміст пояснювальної записки: : 1) аналіз предметної області; 2) огляд існуючих програмних рішень; 3) постановка задачі; 4) опис основних положень і математичних моделей, що використовуються; 5) реалізація програмного рішення.

Дата видачі завдання "_____"_____2020 р.

Керівник випускної роботи _____ Лаврик Т.В.

Завдання приняв до виконання _____ Шелест С.М.

РЕФЕРАТ

Записка: 49 стор., 7 рис., 2 табл., 3 додатки, 9 джерел.

Об’єкт дослідження — комплекс симуляції бойової обстановки.

Мета роботи — розробка захищеної серверної частини програмного комплексу симуляції бойової обстановки.

Методи дослідження — метод аналітичного огляду, метод порівняння та аналогій, метод моделювання, методи розробки, що базуються на мові програмування C#.

Результати — розроблено алгоритм та програмне забезпечення серверної частини програмного комплексу симуляції бойової обстановки. Розроблений алгоритм реалізовано у формі програмного забезпечення, створеного за допомогою інструментального програмного середовища C#.

СИСТЕМА ОБМІНУ ДАНИХ, НАВЧАЛЬНА МАТРИЦЯ, МЕТОД ФУНКЦІОНАЛЬНО-СТАТИСТИЧНИХ ВИПРОБУВАНЬ, АВТОМАТИЗАЦІЯ ОБМІНУ ДАНИХ, СИМУЛЯЦІЯ БОЙОВИХ ДІЙ

ЗМІСТ

ВСТУП

У даний час спостерігається активний розвиток хмарних технологій. Все більше програмного забезпечення стає доступним і пов'язаним з глобальними мережевими платформами та сервісами.

Комп'ютерні ігри за час свого існування зазнали безліч якісних змін в частині виразності і технологічності. Спостерігається проникнення комп'ютерних ігор в різні сфери діяльності – освіту, мистецтво, спорт. В освітній сфері, наприклад, вони можуть виступати інструментом для проведення лекцій, іспитів, професійної практики, також ігри можуть допомогти в таких речах, як розвиток просторового мислення, поліпшення зорової реакції, зняття стресу, тренування організаторських навичок.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Аналітичний огляд платформ для формування каркасу.

Розробка – це детальне проектування з дослідженням і реалізацією: мета, завдання, концепція, комунікації, архітектура, оцінка ресурсів. Кожен проект, який має мультиплеєр повинен мати сервер.

В цілому сервер це одна із найголовніших елментів кліенто-серверного додатку. На даний час є декілька базових підходів для створення серверу (табл. 1.1).

Таблиця 1.1 – Підходи для реалізації

Найменування	**Опис**
На базі готової платформи	Використання готової платформи для створення серверної частини
С інтеграцією елементів готової платформи	Вибіркова інтеграція елементів готової платформи для спрощення або модифікації розробки сервера
Самостійно створення серверної платформи	Створення серверної платформи самостійно для певної задачі

Для візуального огляду ми проведемо аналітику декількох існуючих платформ. Проаналізуємо їх плюси та мінуси. На сьогоднішній день існують різні платформи (табл. 1.2).

Таблиця 1.2 – Платформи для створення серверу

Найменування
Spring Framework
Apache Spark
Photone Network

Починаючи з пункту 1.2, ми детальніше розглянемо вище вказані платформи, щоб підкреслити плюси та мінуси використання цих систем.

1.2 Аналітика існуючих платформ

Photone Network

Photon Unity Networking (PUN) – це пакет Unity для багатокористувацьких ігор. Гнучка система переводить ваших гравців у кімнати, де об’єкти можна синхронізувати по мережі. RPC, власні властивості або події фотонів "низького рівня" - лише деякі функції. Швидке та (необов'язково) надійне спілкування здійснюється через спеціалізовані сервери Photon (ів), тому клієнтам не потрібно підключати один до одного.

Зазвичай, вам не потрібно пам’ятати про структуру пакету PUN, але лише для огляду, ось, як все складно. Пакет PUN охоплює три шари API:

– Найвищий рівень - код PUN, який реалізує особливості Unity, такі як мережеві об'єкти, RPC тощо.

– Другий рівень містить логіку роботи з Photon-серверами, робити сватання, зворотній зв'язок тощо. Це API Realtime. Це вже можна використовувати самостійно. Ви помітите багато перекриття тем між PUN та API реального часу (API.a.a. LoadBalancing API), але це добре.

– Найнижчий рівень складається з файлів DLL, які містять де / серіалізацію, протоколи тощо. На даний час є 2 версії (платна та безкоштовна).

Плюси використання даної платформи:

- Оптимізація роботи с мережевим з’єднанням.
- Широкий асортимент функціоналу на базі готових рішень.
- Стабільність.
- Гнучка система моніторингу.
- Гарна документація.

Недоліки:

- Плата за використання ліцензії.
- Малий асортимент безкоштовної версії.
- Відсутня підтримка для безкоштовної версії.
- Відсутні приклади використання.
- Обмеження при передачі даних.

Spring Framework

Spring Framework — це програмний каркас з відкритим кодом та контейнери з підтримкою інверсії управління для платформи Java.

Основні особливості Spring Framework можуть бути використані будь-яким додатком Java, але є розширення для створення веб-додатків на платформі Java EE. Незважаючи на це, Spring Framework не нав'язує якоїсь конкретної моделі програмування, Spring Framework став популярним в спільноті Java як альтернатива, або навіть доповнення моделі Enterprise JavaBean (EJB).

Spring забезпечує вирішення багатьох завдань, з якими стикаються Java-розробники і організації, які хочуть створити інформаційну систему, засновану на платформі Java.

Через широку функціональності важко визначити найбільш значущі структурні елементи, з яких він складається.

Spring не повністю пов'язаний з платформою Java Enterprise, незважаючи на його масштабну інтеграцію з нею, що є важливою причиною його популярності.

Spring, ймовірно, найбільш відомий як джерело розширень (features), потрібних для ефективної розробки складних бізнес-додатків поза великовагових програмних моделей, які історично були домінуючими в промисловості.

Ще одне його достоїнство в тому, що він ввів раніше невикористовувані функціональні можливості в сьогоднішні панівні методи розробки, навіть поза платформи Java.

Цей фреймворк пропонує послідовну модель і робить її придатною до більшості типів додатків, які вже створені на основі платформи Java. Вважається, що Spring реалізує модель розробки, засновану на кращих стандартах індустрії, і робить її доступною в багатьох областях Java.

Серед можливостей нового фреймворка:

- декларативна підтримка конфігурації HBase;

- спеціалізована підтримка Spring Batch для розробки потокових рішень, які включають операції HDFS і всі типи завдань Hadoop;
- підтримка використання з інтеграцією Spring.

Плюси використання даної платформи:

- Досить гарна підтримка ком’юніті.
- Широкий асортимент функціоналу на базі готових рішень.
- Швидкість роботи.
- Гарна система дебагу.
- Широкі можливості в використанні.
- Підходить під проекти де потрібен обмін даних серед багатьох кліентів.
- Проста система налаштування.
- Широка документація.
- Велика кількість готових модулів для різних рішень.

Недоліки:

- Плата за використання ліцензії.
- Об’ємність передачі даних.
- Відсутність можливостей для сереалізації певних мережевих структур.
- Підходить під проекти де більшість елементів статичні.
- Набір для розробки тільки під певну мову програмування.
- Потреба в великому обсязі серверних ресурсів.
- Нема можливості передачі голосових типу даних.
- Нема можливості передавати файли великого розміру.

Apache Spark

Apache Spark - фреймворк з відкритим вихідним кодом для реалізації розподіленої обробки неструктурованих і слабоструктурованих даних, що входить в екосистему проектів Hadoop.

На відміну від класичного обробника з ядра Hadoop, що реалізує дворівневу концепцію MapReduce з дисковим сховищем, Spark використовує спеціалізовані примітиви для рекуррентной обробки в оперативній пам'яті, завдяки чому дозволяє отримувати значний виграш в швидкості роботи для деяких класів задач, зокрема, можливість багаторазового доступу до завантажених в пам'ять призначених для користувача даних робить бібліотеку привабливою для алгоритмів машинного навчання.

Проект надає програмні інтерфейси для мов Java, Scala, Python, R. Спочатку написаний на Scala, згодом додана істотна частина коду на Java для надання можливості написання програм безпосередньо на Java. Складається з ядра і кількох розширень, таких як Spark SQL (дозволяє виконувати SQL-запити над даними), Spark Streaming (надбудова для обробки потокових даних), Spark MLlib (набір бібліотек машинного навчання), GraphX (призначене для розподіленої обробки графів).

Може працювати як в середовищі кластера Hadoop під керуванням YARN, так і без компонентів ядра Hadoop, підтримує кілька розподілених систем зберігання - HDFS, OpenStack Swift, NoSQL-СУБД Cassandra, Amazon S3.

Плюси використання даної платформи:

- Досить гарна підтримка ком’юніті.
- Широкий асортимент функціоналу на базі готових рішень.
- Швидкість роботи.
- Широкі можливості в використанні.
- Підходить під проекти де потрібен обмін даних серед багатьох клієнтів.
- Проста система налаштування.
- Гарна документація.
- Велика кількість готових модулів для різних рішень.
- Оптимізація під кластерну систему.

Недоліки:

- Система є опенсорс, але це дає можливість знайти слабкі місця, що призводять до збоїв.
- Об’ємність передачі даних.
- Відсутність можливостей для сереалізації певних мережевих структур.
- Підходить під проекти де більшість елементів статичні.
- Потреба в великому обсязі серверних ресурсів.
- Нема можливості передачі голосових типу даних.
- Потребую багато часу для проектування.
- Трафік не захисту.

1.3 Постановка задачі

Сервер — програма, що надає деякі послуги іншим програмам (клієнтам). Зв'язок між клієнтом і сервером зазвичай здійснюється за допомогою передачі повідомлень, часто через мережу, і використовує певний протокол для кодування запитів клієнта і відповідей сервера. Серверні програми можуть бути встановлені як на серверному, так і на персональному комп'ютері, щоразу вони забезпечують виконання певних служб.

У даній роботі потрібно реалізувати консольне керування для взаємодії клієнта з серверною частиною. Потрібно створити також систему API і систему розрахунку балістики.

Консольне керування

Розробка зручної системи керування серверу за допомогою командного рядку. Консольне керування повинно забезпечувати простоту та інформативність для користувача. Набір фунціоналу має бути достатнім, щоб користувач мав змогу легко керувати процесом роботи серверу та здійснювати маніпуляцію даних, якими оперує.

API

На сервері також потрібно реалізувати систему API. Це одна з необхідних частин проекту, що дає можливість іншим розробникам швидко створювати

додатки до існуючого процесу. Система API – повинна бути простою в реалізації та мати широкий функціонал для різних потреб.

Система розрахунку балістики

Потрібно створити систему розрахунку балістики для різних типів снарядів за різних погодних умов. Частина системи буде реалізована в API для зручності використання та розширення можливостей для розробників.

2 МАТЕМАТИЧНА МОДЕЛЬ БАЛІСТИКИ

2.1 Теоретичний опис моделі

Існуючий спосіб визначення установок для стрільби реактивними системами залпового вогню (РСЗВ) на основі повної підготовки полягає у розрахунку поправок на відхилення метеорологічних, балістичних і геофізичних умов стрільби від табличних значень. Метеорологічні умови визначаються за даними бюлетеня «Метеосередній» або «Метеонаближений», термін придатності якого може складати від 2 годин і більше, в результаті чого точність розрахунку поправок на умови стрільби зменшуються. Спосіб визначення установок для стрільби РСЗВ, що пропонується, будується по-перше, на обліку метеорологічних факторів на трьох ділянках траєкторії: активна ділянка траєкторії (АДТ), пасивна ділянка траєкторії (ПДТ) і ділянка розльоту бойових елементів (ДРБЕ). По-друге, поправки на умови стрільби визначаються за допомогою Таблиць стрільби даних бюлетеня «Метеосередній».

Такий спосіб є важким для проведення розрахунків оскільки потребує значного часу. та вимагає враховувати двадцять метеорологічних виправлень. Аналіз останіх досліджень і публікацій. Метеорологічна підготовка стрільби в артилерійських підрозділах включає: організацію прийому метеорологічних бюлетенів «Метеосередній» від метеорологічних станцій, а при неможливості прийому, одержання їх від старшого штабу.

Визначення установок для стрільби здійснюється за допомогою Таблиць стрільби. Для цього визначаються метеорологічні умови: відхилення наземного тиску атмосфери(ΔНо) на висоті вогневої (стартової) позиції, балістичного відхилення температури повітря (ΔТ) в межах повної траєкторії, балістичного відхилення температури повітря (ΔТа) в межах активної ділянки траєкторії, подовжньої і бокової складових балістичного вітру (Wax, Waz) в межах активної і пасивної (Wnx, Wnz) ділянок траєкторії, а також на ділянці розльоту бойових елементів (Wex, Wez).

Метеорологічні бюлетені «Метеосередній» мі стять усі необхідні дані для визначення метеорологічних умов, що враховуються в межах ПДТ і АДТ. Під

час стрільби на великі дальності (до 70 км) в умовах рівнинної місцевості дані бюлетеня «Метеосередній» забезпечують визначення вітру на ділянці розльоту бойових елементів. Траєкторія руху реактивного снаряду поділяють на три ділянки (рис. 2.1):

1 – активна ділянка траєкторії (АДТ);

2 – пасивна ділянка траєкторії (ПДТ);

3 – ділянка розльоту бойових елементів (ДРБЕ).

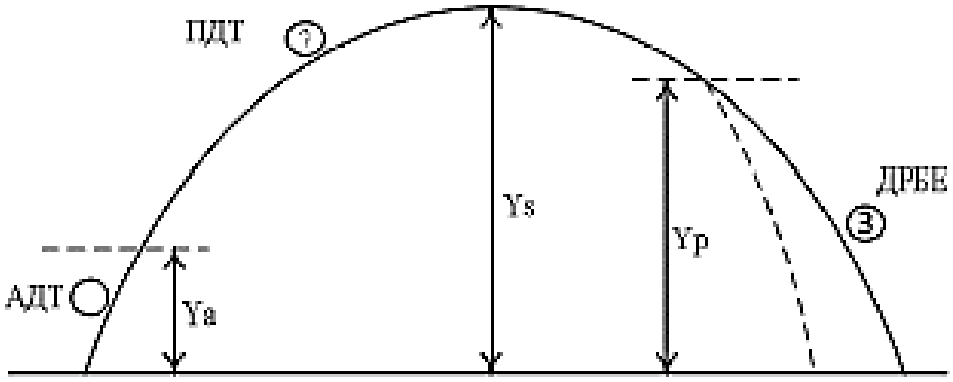

Рисунок 2.1 – Траєкторія руху реактивного снаряду

Під час визначення вирахуваних установок для стрільби обчислюють поправки в приціл та в доворот від основного напрямку на відхилення метеорологічних умов від табличних значень.

Під час визначення поправки в приціл необхідно визначати поправочні коефіцієнти: на подовжню складову балістичного вітру відповідно в межах АУТ, ПДА і ділянки розльоту бойових елементів; на відхилення наземного тиску атмосфери від табличного значення; на балістичне відхилення температури повітря; на спільний вплив відхилень температури і наземного тиску повітря. Під час визначення поправки в доворот від основного напрямку ураховують поправочні коефіцієнти на бокову складову балістичного вітру в межах АДТ, ПДТ і ділянки розльоту бойових елементів відповідно. У даний спосіб система поправок характеризується роздільним обліком вітру в межах АДТ, ПДТ і ділянки розльоту бойових елементів.

Такий підхід до обліку метеорологічних факторів, і зокрема вітру, не є єдино можливим і остаточно прийнятим. Можлива й інша система

метеорологічних виправлень, особливо це стосується порядку обліку вітру, про що буде сказано нижче. При існуючому способі доводиться враховувати понад двадцять метеорологічних виправлень.

Розрахунок такої великої кількості балістичних та метеорологічних величин і відповідних їм виправлень без застосування ЕОМ вимагає значного часу. Проведений аналіз поправочних коефіцієнтів, що характеризують відхилення снаряда за дальністю та напрямком від табличної точки падіння в межах АДТ, ПДТ, ДРБЕ показує, що вплив подовжнього вітру на цих ділянках приблизно однаковий, за винятком дальностей стрільби від 30 км до 40 км, де вплив вітру на ПДТ в 2…2,5 рази менший в порівнянні з його впливом у межах АДТ і на ділянці розльоту бойових елементів. Вплив бокового вітру в межах АДТ і ПДТ ідентичний на всіх дальностях стрільби, а на ділянці розльоту бойових елементів він в 2…3 рази перевершує вплив на АДТ і ПДТ тільки на дальностях 30…40 км. Таким чином, можна зробити висновок про необхідність більш точного визначення й обліку вітру на цих трьох ділянках траєкторії.

Отримані результати дозволяють говорити про можливість обліку вітру в межах усієї траєкторії, без поділу її на окремі ділянки. Це дозволило б скоротити кількість метеорологічних величин, що обчислюються, та спростити систему метеорологічних виправлень. Дослідження величини відхилення снаряда від розрахункової табличної точки падіння через неврахування впливу метеорологічних величин показує, що вона може досягати кілька сотень метрів. Цілком очевидно, що за умови таких відхилень снарядів вогневе завдання може бути невиконаним. Існуючий спосіб визначення установок для стрільби реактивними снарядами потребує розрахунку вагових коефіцієнтів для обліку балістичних відхилень температури повітря і балістичного вітру.

Апроксимація функцій реального закону розподілу заміняється лінійним законом, що являться наближеним. У зв’язку з чим розрахунок установок стає громіздким і вимагає багато часу, що у сучасному швидкоплинному бою може привести до невиконання бойового завдання. Постановка завдання. Як бачимо в ускладненому способі визначення установок для розрахунку поправок на

відхилення метеорологічних умов стрільби від табличних значень необхідно розрахувати вагові функції впливу температури повітря та вітру для АДТ, ПДТ, ДРБЄ і визначити за ними вагові коефіцієнти для кожного типу ЛА, що являє собою громіздкий процес, а заміна дійсних законів розподілу лінійними призводить до значних помилок в розрахунку поправок (5-7 % дальності) [2, 3]. В способі, що пропонується, немає необхідності приводити середні значення метеорологічних елементів до балістичних.

Цю задачу вирішує система диференційних рівнянь (СДР) в яку вводять дійсні балістичні параметри конкретного ЛА, крім того, за допомогою системи диференційних рівнянь враховується взаємовплив і нелінійність збуджуючих факторів [3]. Метою розробки запропонованого способу є вдосконалення способу визначення установок для стрільби (пусків) РСЗВ за допомогою системи диференціальних рівнянь та за рахунок автоматизації процесу отримання метеорологічних даних від метеорологічного комплексу з'єднаного з артилерійським обчислювачем.

Даний спосіб, у порівнянні з існуючим, має забезпечити необхідну швидкість і достовірність визначення установок з урахуванням метеорологічних, балістичних і геофізичних умов стрільби та балістичних параметрів руху ЛА, що дозволить підвищити точність та скоротити час визначення установок для стрільби (пусків) РСЗВ.

2.2 Математична реалізація

Для розв'язання системи диференційованих рівнянь запропоновано використовувати метод Рунге-Кутта четвертого порядку (2.1).

$$
\begin{cases}
\dot{x} = V\cos\theta\cos\psi\ /\left(1-2y/R_з\right);\\
\dot{y} = V\sin\theta;\\
\dot{z} = V\cos\theta\sin\psi;\\
\dot{V} = a_p - a_x\cos\gamma - g_0\sin\theta\left(1-2y/R_з\right);\\
\dot{\theta} = -\dfrac{\cos\theta g_0\left(1-2y/R_з\right)}{V} - \dfrac{a_x\cos\gamma\, W_x\ \sin\theta}{V\,V_r} +\\
+V\cos\theta/\left(R_з + y\right) - \Omega_з\cos B\sin(a_Г - \psi);\\
\dot{\psi} = -\dfrac{a_x\cos\gamma\, W_z}{\cos\theta V\, V_r} + \Omega_з\left(\sin B - \cos B\cos(a_Г - \psi)\,\mathrm{tg}\theta\right);\\
\dot{\pi}(y) = -\pi(y)\dot{y}/\left(R\left[\tau_y + \Delta\tau\right]\right).
\end{cases}
\qquad (2.1)
$$

де – поточне значення швидкості змін координат ЛА, м/с; V& – прискорення ЛА, м/с2 ; θ & , ψ& – швидкість зміни кута кидання і напрямку ЛА, рад/с; π(y) – функція розподілу тиску атмосфери з висотою, мм. рт. ст; Г а – азимут пуску, рад; В – широта стартової позиції, рад; Ωз – кутова швидкість обертання Землі, рад/с; Rз – радіус Землі, м; R – газова постійна для повітря, кгм/кг·град; t – польотний час ЛА, с; Δτ – відхилення віртуальної температури повітря від табличної, град; V– швидкість ЛА, м/с; θ – кут кидання, рад; ψ – кут рискання, рад; р а – реактивне прискорення, м/с2:

$$
a_p = \frac{\omega_0\left(I_{1N} + K_1\Delta Tзр\right)}{m_0\left[\tau_{aN} - K_2\Delta Tзр\right]\times\left(1-\mu_y\right)}, \qquad (2.2)
$$

де К1 , К2 – коефіцієнти, що враховують вплив температури реактивного заряду на одиничний імпульс тяги та час роботи реактивного двигуна відповідно кг·сек/кг; сек/град; I1N – табличне значення одиничного імпульсу тяги кг·сек/кг; τаN – табличний час роботи реактивного двигуна, с; ω0 – вага реактивного заряду, кг. Коефіцієнт витрати палива (μу), кг/с:

$$
\mu_y = \frac{\omega_0(t - t_H)}{g_0 m_0\left(\tau_{aN} + K_2\Delta Tзр\right)}, \qquad (2.3)
$$

де tН – час вмикання реактивного двигуна, с; t – польотний час ЛА, с; 0 g – прискорення вільного падіння, м/с2 . маса ЛА (m0), кг·с 2 /м:

$$
m_0 = q_0 / g_0\ , \qquad (2.4)
$$

відхилення температури заряду від табличного значення (ΔTзр) град. :

$$\Delta Tзр = Tзр - 15^0, \quad (2.5)$$

прискорення сили лобового опору (x a), м/с2 :

$$a_x = 0.474 \frac{id^2}{q_0 + \Delta q} \pi(y) V_{r\tau}{}^2 C_x(V_{r\tau}), \quad (2.6)$$

де 0 q – повна вага ЛА, кг; Δq – відхилення ваги ЛА від табличного значення, кг; i – коефіцієнт форми ЛА; d – калібр, м; коефіцієнт вітру (γ):

$$\cos\gamma = \frac{V - W_x \cos\theta}{V_r}, \quad (2.7)$$

відносна швидкість обертаємих ЛА (r V): (2.8)

$$V_r = V \times \sqrt{1 - \frac{2\,(W_X \cos q \cos y + W_Z \sin y \cos q)}{V} + \frac{W^2}{V^2}}; \quad (2.8)$$

відносна швидкість ЛА з обліком температури повітря (Vrτ), м/с:

$$V_{rt} = V_r \sqrt{\tau_{ON} / (\tau_y + \Delta\tau)}, \quad (2.9)$$

де τON – табличне значення віртуальної температури повітря на Землі, град; Vr – відносна швидкість ЛА;

розподіл віртуальної температури з висотою (τy), град.:

$$\tau_y = \begin{cases} 289.0 - 0.006328\,Y \text{ при } 0 \le Y \le 9324; \\ 230 - 0.006328(Y - 9324) + 0.000001172 \times \\ \times (Y - 9324)^2 \text{ при } 9324 < Y \le 12000; \\ 221.5 \text{ при } Y > 12000. \end{cases} \quad (2.10)$$

Активна ділянка траєкторії для оперених ЛА: прискорення сили лобового опору (x a):

$$a_x = 0.474 \frac{i_a d^2}{q_0} 10^3 \frac{\tau_{ON}}{\tau_y + \Delta\tau} \frac{\pi(y) F_{58}(V_{r\tau})}{(1 - \mu_y)}. \quad (2.11)$$

Пасивна ділянка траєкторії для оперених ЛА: прискорення сили лобового опору (x a):

$$a_x = 0.474 \frac{i_П d^2}{q_П} 10^3 \frac{\tau_{ON}}{\tau_y + \Delta\tau} F_{58}\left(V_{rr}\right). \quad (2.12)$$

Ділянка розкриття бойових елементів для оперених ЛА: прискорення сили лобового опору (x a):

$$a_x = 0.474 \frac{i_{БЕ} d^2}{q_{БЕ}} 10^3 \frac{\tau_{ON}}{\tau_y + \Delta\tau} F_{58}\left(V_{rr}\right), \quad (2.13)$$

де i_a, $i_П$, $i_{БЕ}$– коефіцієнти форми, отримані шляхомрішення СДУ з використанням ТС РКЗВ; $F_{58}(V_{rt})$ – закон опору повітря.

Вплив вітру на активній ділянці траєкторії для оперених ЛА: 1-й випадок: θ → θ0 + δθ; ψ = ψ0 + Δψ:

$$V_r = V\sqrt{\begin{array}{l}1 - (2W_{ax}(\cos\theta\ \cos\psi\ + \gamma_w W_{ax} \times \\ \times \sin\psi\ - \gamma_м W_{az} \sin\theta\ \cos\psi\ + \gamma_w\gamma_М \times \\ \times W_{ax} W_{az} tg\theta\ \sin\psi) - 2W_{az}(\cos\theta\ \sin\psi + \\ + \gamma_w W_{ax} \cos\psi\ - \gamma_м W_{az} \sin\theta\ \sin\psi\ - \\ - \gamma_w\gamma_м W_{ax} W_{az} tg\theta\ \cos\psi))\ /\ V + W_a^2/V^2;\end{array}} \quad (2.14)$$

3-й випадок: 0 0 θ → θ + δθ ψ → ψ Δψ ; - :

$$V_r = V\sqrt{\begin{array}{l}1 - (2W_{ax}(\cos\theta\ \cos\psi\ + \gamma_м W_{az} \times \sin\theta\ \cos\psi - \gamma_w W_{ax} \sin\psi - \gamma_w\gamma_м \times W_{ax} W_{az} tg\theta\ \sin\psi) - \\ -2W_{az}(\cos\theta\ \sin\psi + \gamma_м W_{az} \sin\theta\ \sin\psi + \gamma_w W_{ax} \cos\psi + \gamma_w\gamma_м W_{ax} W_{az} tg\theta\ \cos\psi))\ /\ V + W_a^2/V^2\end{array}}. \quad (2.15)$$

4-й випадок: 0 0 - ; · θ → θ δθ ψ → ψ + Δψ :

$$V_r = V\sqrt{\begin{array}{l}1 - (2W_{ax}(\cos\theta\ \cos\psi\ + \gamma_w W_{ax} \times \\ \times \sin\psi\ + \gamma_м W_{az} \sin\theta\ \cos\psi\ + \gamma_W\gamma_М \times \\ \times W_{ax} W_{az} tg\theta\ \sin\psi) - 2W_{az}(\cos\theta\ \sin\psi\ - \\ - \gamma_w W_{ax} \cos\psi\ + \gamma_м W_{az} \sin\theta\ \sin\psi\ - \\ - \gamma_w\gamma_м W_{ax} W_{az} tg\theta\ \cos\psi))\ /\ V\ + W_a^2/V^2\end{array}} \quad (2.16)$$

де Y_M, Y_W,, – вітрові коефіцієнти прямого і перехресного впливу вітру на оперенні реактивні снаряди визначаються опитним шляхом і розміщуються в Таблицях стрільби. Вплив вітру на пасивній ділянці траєкторії для оперених ЛА визначаються за залежностями:

$$V_г = V\sqrt{1 - 2\,(W_{пх}\cos\theta\,\cos\psi + W_{пz}\cos\theta\,\sin\psi)\,/\,V + W_п^2/V^2}\,,$$
$$W_{пх} = W_п\,\cos\alpha w_п;\quad W_{пz} = W_п\,\sin\alpha w_п;$$
$$W_п^2 = W_{пх}^2 + W_{пz}^2; \qquad (2.17)$$

де W , w п п α – середнє значення швидкості вітру і дирекційного кута вітру в шарі атмосфери; $W_{ПХ}$, $W_{ПZ}$, – поздовжня та бокова складові балістичного вітру.

Вплив вітру на ділянки розльоту бойових елементів:

$$V_г = V\sqrt{\begin{array}{l}1 - (2W_{БЕx}(\cos\theta\,\cos\psi + \gamma_м W_{БЕx}\times \\ \times\sin\psi - \gamma_м W_{БЕz}\,\sin\theta\,\cos\psi - \gamma_м\gamma_м\times \\ \times W_{БЕx}W_{БЕz}tg\theta\,\cos\psi) - 2W_{БЕx}(\cos\theta\,\sin\psi - \\ - \gamma_м W_{БЕx}\cos\psi - \gamma_м W_{БЕz}\sin\theta\,\sin\psi + \\ + \gamma_м\gamma_м W_{БЕx}W_{БЕz}tg\theta\,\cos\psi))\,/\,V + W_{БЕ}^2/V^2\end{array}}$$

де $W_{БЕx} = W_{БЕ}\cos\alpha w_{БЕ};\quad W_{БЕz} = W_{БЕ}\sin\alpha w_{БЕ};$

$$W_{БЕ}^2 = W_{БЕx}^2 + W_{БЕz}^2; \qquad (2.18)$$

γМ – коефіцієнт впливу вітру на бойові елементи на ДРБЕ визначаються дослідним шляхом. $W_{БЕ}$, $\alpha w_{БЕ}$ – середнє значення швидкості вітру і дирекційного кута вітру в шарі атмосфери де летить ЛА; $W_{БЕx}$, $W_{БЕz}$ – поздовжня та бокова складові балістичного вітру на ДРБЕ. Використання способу, що пропонується, у сукупності з усіма істотними ознаками, дозволяє розрахувати установки для кожної стрільби за лічені секунди (2…5 с) після введення даних, які надходять з метеорологічної станції та вогневої позиції на момент стрільби.

Даний спосіб виключає необхідність розробки вагових коефіцієнтів за температурою та вітром, розрахунку поправок на відхилення метеорологічних, балістичних і геофізичних умов стрільби від табличних значень, тому що ці умови враховуються автоматично в ході рішення СДР. Таким чином, під час

використання способу, що пропонується значно підвищується точність підготовки установок для стрільби реактивними снарядами. Так серединна помилка підготовки установок складає 0,8…0,9 % в порівнянні з існуючим способом 1,3…1,5 %.

Автоматизація процесів отримання даних про умови стрільби, розрахунку вирахуваних установок та передачі їх на реактивні установки дозволить значно скоротити час на підготовку установок до 2...5 с в порівнянні з 24 хвилинами за умови аналітичного їх визначення.

Запропонований спосіб визначення установок для стрільби реактивними снарядами шляхом рішення системи диференційних рівнянь руху ЛА дозволяє значно підвищити точність підготовки ударів і в рази скоротити час на підготовку установок.

Методика урахування метеорологічних, балістичних і геодезичних факторів може бути використана для різних систем залпового вогню.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО РІШЕННЯ

Балістична модель відіграє досить важливу роль. Для даної моделі була створена програмна реалізація. Дана реалізація дозволяє виконувати розрахунок на сервері на основі реальних даних, які ми отримуємо від ігрових клієнтів. Програмна реалізація даної системи наведена у Додатку В.

Для розробки сервера був обраний набір пакетних рішень C#. Для простити використання та розширення функціоналу було вирішено використовувати елементи готової платформи Photone Network (безкоштовної версії). Деякі елементи даної системи досить гарно підходять під цілі проекту та зменшать час розробки.

3.1 Консольне керування

Для розробки консольного інтерфейсу керування використовували стандартні можливості C#. Багато необхідних речей вже вбудовані, тому розроблення інтерфейсу в ньому досить просте завдання, якщо розумієш, як що працює.

Головне меню консольного керування зображено на (рис. 3.1).

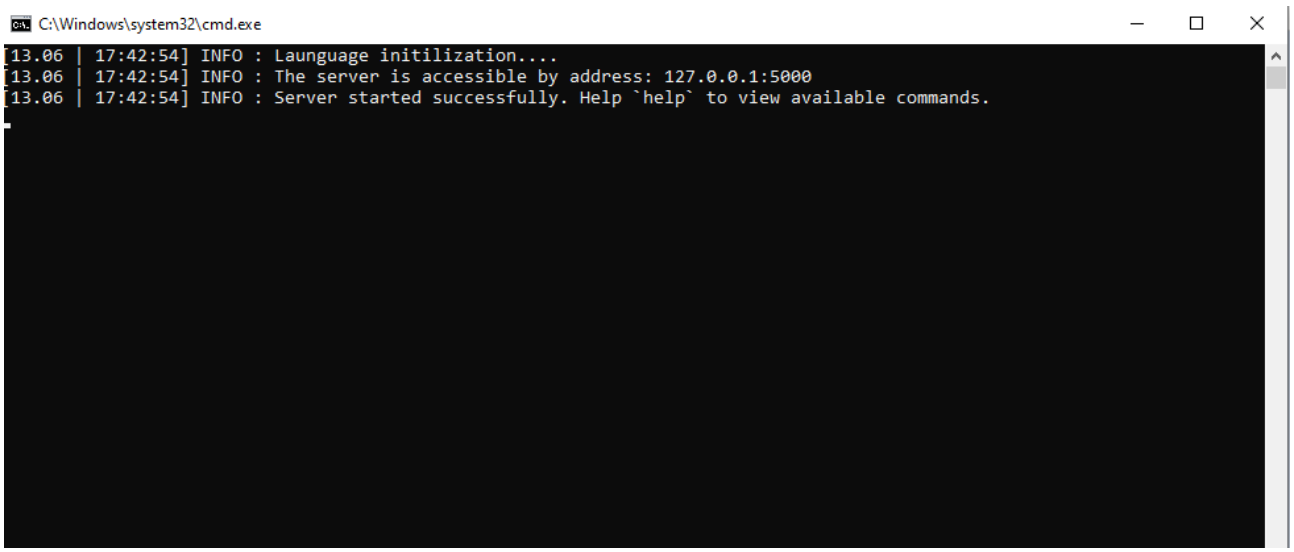

Рисунок 3.1 – Головне меню

В даному вікні ми можемо бачити:

- Гарну інформаційну систему за часом (рис. 3.2).
- Тільки необхідний обьем інформації про поточні події.
- Можливість стоверння та введення команд (рис. 3.3).
- Елементи локалізації (рис. 3.4).

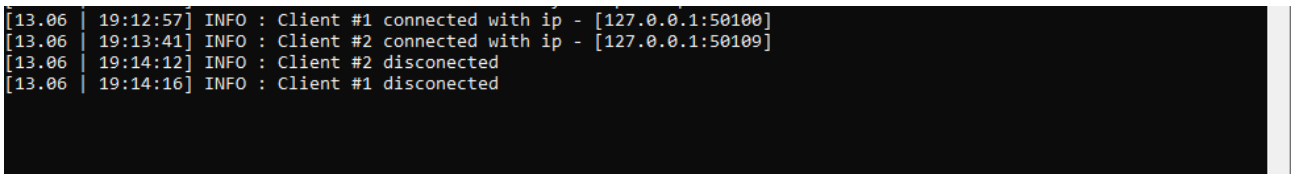

Рисунок 3.2 – Зображення системи виводу

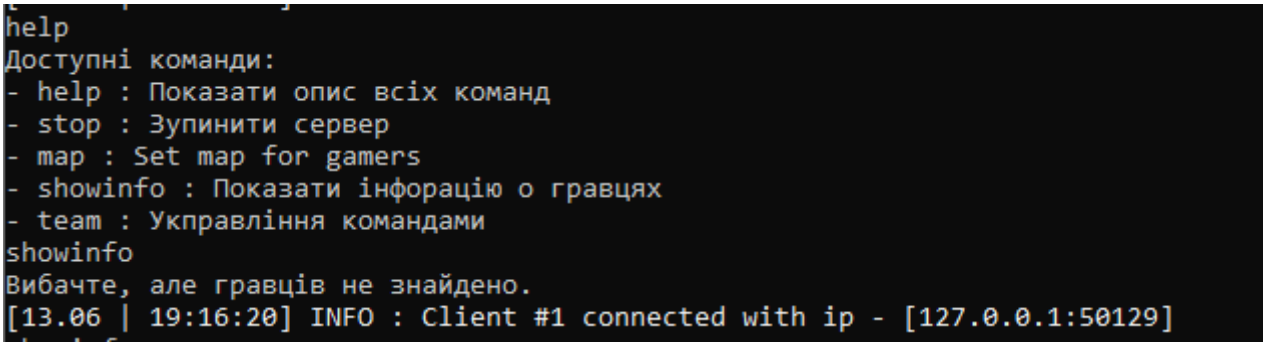

Рисунок 3.3 – Вивід інформації про доступні команди

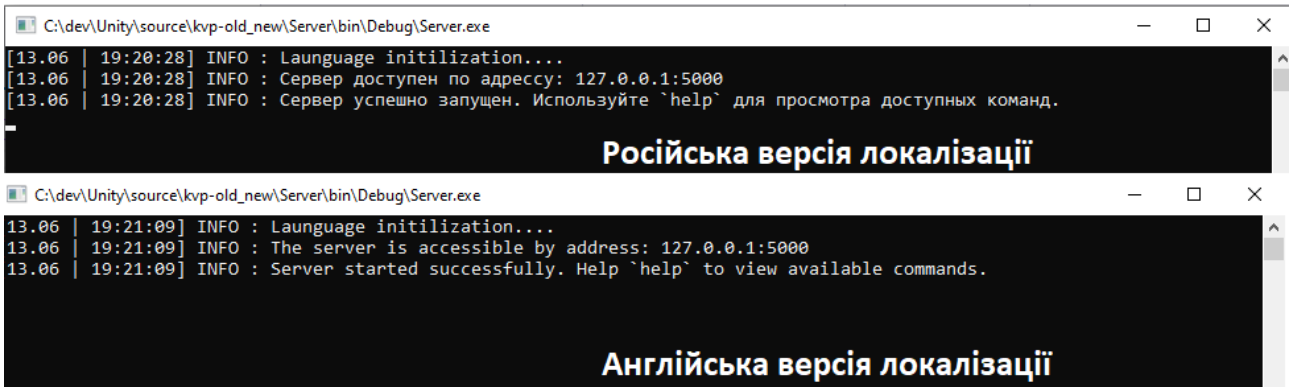

Рисунок 3.4 – Приклади роботи сервера при різній мові

У цьому додатку ми можемо бачити Головний та Підлеглі класи. Основний алгоритм роботи консольного керування:

- Ініціалізуємо менеджер команд.
- Створюємо клас, який наслідує абстрактний клас Command для реалізації команди.
- Реєструємо наш клас в менеджері.
- Надалі після запуску програми наша команда буде вже доступна для користування.

Реалізація програмного коду для локалізації нашої системи наведенп в Додатку А.2.

У цьому додатку ми можемо бачити Головний та Підлеглі класи. Основний алгоритм роботи системи локалізації:

- Ініціалізуємо менеджер локалізації.

- Створюємо клас, який наслідує абстрактний клас LanguageFile для реалізації команди.
- Реєструємо наш клас в менеджері.
- Вказуємо тип мови
- Надалі після запуску програми наша команда буде вже доступна для користування.

3.2 API для розробників

Для розробки API буде використовуватися стандартний набір функціоналу мови програмування C#. Розробка цього елменту досить важлива і займає більшу половину часу по створенню проекту.

Набір API включає досить великий набір функціоналу:

- Логування.
- Система розподілення на команди.
- Система гравців.
- Система мережевого зв’язку.
- Система конвертації та захисту даних.
- Система подій та їх слухачів.

Система логування — це створення файлів це файлів, що містять системну інформацію роботи сервера або комп'ютера, в які заносяться певні дії користувача або програми. Іноді також вживається російськомовний аналог поняття - журнал.

Система логування на сервері має наступний вигляд (рис. 3.5-3.6).

Рисунок 3.5 – Ірархія папок логування

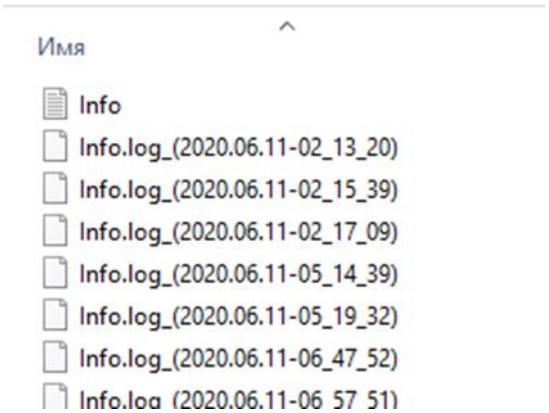

Рисунок 3.6 – Фали логів

Їх призначення - протоколювання операцій, що виконуються на машині, для подальшого аналізу адміністратором. Регулярний перегляд журналів дозволить визначити помилки в роботі системи в цілому, конкретного сервісу або сайту (особливо приховані помилки, які не виводяться при перегляді в браузері), діагностувати зловмисну активність, зібрати статистику відвідувань сайту.

Система розподілення на команди – це розподілений набір ролей, які допомагають гравцю відчути приналежність до система. Розподілення гравців на команди допомагають розробнику встановили певні задачі та надати особливі можливості для кожного з типів.

Система гравців – це система, що зберігає в собі персональну інформацію про гравця. Ця система допомагає наладити комунікацію між гравцями, щоб кожен гравець міг бачити з ким він спілкується. Також дана система допомагає в розробці, адже кожен гравець має певний ідентифікатор, що дозволяє направляти.

Система мережевого зв’язку - набір угод інтерфейсу логічного рівня, які визначають обмін даними між різними програмами. Ці угоди задають однаковий спосіб передачі повідомлень і обробки помилок. Сигнальний протокол використовується для управління з'єднанням - наприклад, установки, переадресації, розриву зв'язку.

Це одна з найголовніших під-систем у нашому сервері. Ця система дозволяє:

- Приймати / Надсилати дані.
- Шифрувати інформацію, яка передається між клієнтом та сервером.
- Сортувати дані.
- Встановлювати пріоритетність.
- Синхронізація всіх даних в системі.

Система конвертації та захисту даних – ця система працює з деяким типом даних, що потребують додаткового захисту. Система шифрує інформацію

за допомогою криптографічних алгоритмів та передає її в мережеву частину для розсилання.

Система подій та їх слухачів – система, що дозволяє більш тонко працювати з подіями, що трапляются на сервері. Дана підсистема дозволяє відокремити та налагодити структуру коду і зробити систему більш продуктивніше.

3.3 Програмна реалізація

Сервер – це база всієї системи. Правильна реалізація забезпечує половину успіху при розробці додатку. Повна програмна реалізація базової системи серверу наведена в Додатках А та Б.

ВИСНОВКИ

Завданням цього проекту була розробка функціонального сервера для симулятора бойових дій. Для реалізації цього завдання були виконані наступні етапи:

- Проаналізовано існуючі рішення.
- Сформовано план дій.
- Виконана реалізація елементів плану.

Головним результатом проведеної роботи є створення функціонуючого додатка, який виконує необхідне коло завдань.

Уся необхідна робота по здійсненню методів управління інформацією та її модифікації в цілісному вигляді прихована всередині і користувачеві не потрібно знати про неї, щоб успішно вирішувати коло завдань, що виникають і пов'язані з використанням інформації.

Даний продукт без сумніву може конкурувати з існуючими на даний час системами по симуляції бойових дій.

Програма інтуїтивно проста і зрозуміла для будь-якого користувача, для її використання не потрібно спеціального навчання.

Система охоплює широкий спектр завдань: обмін інформацією між клієнтами, розрахунок пострілів гармат, обробка інформації користувачів, обробка інформації про команди, і головне, захист даних.

СПИСОК ЛІТЕРАТУРИ

1. Подготовка стрельби и управлення огнем артилерии. – М.: Военное издательство СССР,1987.– 420 с.

2. Оцінка точності урахування метеорологічних факторів при стрільбі на великі дальност /: Макеєв В.І. та ін. // Збірник наукових праць Харківського університету Повітряних Сил. – Х.: ХУПС, 2010. – №3 (23). – С. 85–89.

3. Макеев В.І. Методика определения поправок на нелинейность и взаимодействие возмущающих факторов / Макеев В.І. // Электронное моделирование. – 2012. – № 1, т. 34.– С. 109-119.

4. Дмитриевский А.А. Внешняя балистика / А.А. Дмитриевский, Л.А. Лысенко. – М.: Машиностроение, 2005. – 608 с.

5. Шапкин С. Компьютерные игры - польза или вред // Психологический журнал, том 20. 1999. №1.

6. Al Sweigart Automate the Boring Stuff with Python: Practical Programming for Total Beginners. - 1st Edition No Starch Press, 2015.

7. Amit Saha Doing Math with Python: Use Programming to Explore Algebra, Statistics, Calculus, and More!. - 1st Edition No Starch Press, 2015.

8 .Learn Version Control with Git // TOWER URL: https://www.gittower.com/learn/git/ebook/en/command-line/introduction (дата обращения: 22.03.2017).

9 .Socket.IO - Docs // Socket.IO URL: https://socket.io/docs/ (дата обращения: 17.01.2017).

Додаток А

Абстрактний клас, що має набір певних параметрів для створення команди.

```
namespace KVP_Api.Commands
{
    public abstract class Command
    {

        private string name;
        private string[] description;

        public Command(string name, string[] description)
        {
            this.name = name;
            this.description = description;
        }

        public string getName()
        {
            return name;
        }

        public string[] getDescription()
        {
            return description;
        }

        public abstract void execute(string[] args);
    }
}
```

Головний клас, що забачує роботу з командами та їх процесами.

```
using System;
using System.Collections.Generic;
using System.Threading;

namespace KVP_Api.Commands
{
    public class CommandManager
    {
        private Dictionary<string, Command> commands;

        public CommandManager()
        {
            commands = new Dictionary<string, Command>();
            new Thread(StartReadCommand).Start();
        }

        private void StartReadCommand()
        {
            string line;

            while ((line = Console.ReadLine()) != null)
            {
                string[] args = line.Split(' ');
                line = args[0];
                args = CopyOfRange(args, 1, args.Length);

                if (commands.ContainsKey(line))
                {
```

```
                commands[line].execute(args);
            }
            else
            {
                Console.WriteLine("Command not found. Type `help` to help");
            }
        }
    }

    private string[] CopyOfRange(string[] src, int start, int end)
    {
        int len = end - start;
        string[] dest = new string[len];
        Array.Copy(src, start, dest, 0, len);
        return dest;
    }

    /**
     * Регистрация команд
     * @param command - Экземпляр класса команды
     */

    public void RegisterCommand(Command command)
    {
        commands.Add(command.getName(), command);
    }


    /**
     * Отключить команду за именем
     * @param name - имя команды
     */
    public void UnRegisterCommand(string name)
    {
        commands.Remove(name);
    }
    public void UnRegisterAllCommands()
    {
        commands.Clear();
    }

    /**
     * Получить все экземпляры класов команд
     * @return
     */
    public ICollection<Command> GetCommands()
    {
        return commands.Values;
    }

    /**
     * Получить экземпляр класса команды за именем
     * @param name - имя команды
     * @return - экземпляр класса команды
     */
    public Command GetCommand(string name)
    {
        return commands[name];
    }
}
}
```

Клас для створення файлу локалізаї

```
using System.Collections.Generic;

namespace KVP_Api.Language
{
    public abstract class LanguageFile
    {
        private LaungeCode laungeCode;
        private Dictionary<string, string> worbs;

        public LanguageFile(LaungeCode laungeCode)
        {
            this.laungeCode = laungeCode;
        }

        public void SetWorbsData(Dictionary<string, string> worbs)
        {
            this.worbs = worbs;
        }

        public string GetMessage(string key)
        {
            return worbs[key];
        }

        public LaungeCode GetLaungeCode()
        {
            return laungeCode;
        }
    }
}
```

Головний клас взаємодії з локалізацією

```
using System.Collections.Generic;

namespace KVP_Api.Language
{
    public class LanguageManager
    {
        private LaungeCode languageType;
        private List<LanguageFile> languages = new List<LanguageFile>();
        private LanguageFile languageFile;

        /*public LanguageManager()
        {
            init();
        }

        private void init()
        {

        }*/


        public void AddLanguage(LanguageFile language)
        {
            languages.Add(language);
        }

        public void SetLanguage(LaungeCode laungeCode)
        {
            foreach(LanguageFile lng in languages)
            {
                if (lng.GetLaungeCode().Equals(laungeCode))
```

```
                {
                    languageFile = lng;
                }
            }
        }

        public LanguageFile GetLanguageFile()
        {
            return languageFile;
        }
    }
}
```

Набір типів мов, які доступні для локалізації

```
namespace KVP_Api.Language
{
    public enum LaungeCode
    {
        EN,
        RU,
        UA,
        NONE
    }
}
```

Ініціалізація модуля мови

```
string lngCode = ConfigurationManager.AppSettings["LANGUAGE"];
switch (lngCode)
{
case "EN":
laungeCode = LaungeCode.EN;
break;
case "RU":
laungeCode = LaungeCode.RU;
break;
case "UA":
laungeCode = LaungeCode.UA;
break;
}
```

Підключення команд для використання в систему керування

```
private void RegisterCommands()
{
commandManager.UnRegisterAllCommands();
commandManager.RegisterCommand(new HelpCommand());
commandManager.RegisterCommand(new StopCommand());
commandManager.RegisterCommand(new MapCommand());
commandManager.RegisterCommand(new ShowInfoCommand());
commandManager.RegisterCommand(new TeamControllCommand());
}
```

Додаток Б

Програмна реалізація системи подій

```
namespace KVP_Api.Events
{
    public class DefaultEvent : IEvent
    {
        public DefaultEvent() { }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Management.Instrumentation;
using System.Text;
using System.Threading.Tasks;

namespace KVP_Api.Events
{
    public class EventManager
    {

        private Dictionary<string, List<Action<IEvent>>> __register;

        public EventManager()
        {
            this.__register = new Dictionary<string, List<Action<IEvent>>>();
        }

        public void On(string name, Action<IEvent> callback)
        {
            if (!this.__register.ContainsKey(name))
            {
                this.__register[name] = new List<Action<IEvent>>();
            }
            this.__register[name].Add(callback);
        }


        public void Trigger(string name)
        {
            this.Trigger(name, new DefaultEvent());
        }

        public void Trigger(string name, IEvent data)
        {
            if (this.__register.ContainsKey(name))
            {
                foreach (Action<IEvent> callback in this.__register[name])
                {
                    callback(data);
                }
            }
        }

    }
}

namespace KVP_Api.Events
{
    public interface IEvent { }
```

```
}
```

Програмна реалізація системи пакування повідомлень

```
namespace KVP_Api.Messages
{
    public class Message
    {
        private string msg = "";
        private MessageType messageType;
        private MessageColor color;

        public Message(string msg)
        {
            this.msg = msg;
        }

        public void SetType(MessageType messageType)
        {
            this.messageType = messageType;
        }

        public void SetColor(MessageColor color)
        {
            this.color = color;
        }

        public string GetText()
        {
            return msg;
        }
    }
}
namespace KVP_Api.Messages
{
    public enum MessageColor
    {
        BLACK,
        BROWN,
        GRAY,
        WHITE,
        YELLOW,
        ORANGE,
        RED,
        PINK,
        PURPLE,
        BLUE,
        GREEN
    }
}
namespace KVP_Api.Messages
{
    public enum MessageType
    {
        SYSTEM,
        CHAT,
        INFO,
        ERROR
    }
}
```

Програмна реалізація мережевої частини

```
using KVP_Api.Network.Documents;
using KVP_Api.Network.Packets;
```

```
using KVP_Api.Network.Utils;
using Newtonsoft.Json;

namespace KVP_Api.Network
{
    public class NetworkManager
    {
        public static void SendData(NetworkConnection networkConnection, int idPacket,
Document data)
        {
            Document document = new Document();
            document.SetData(data.GetData());

            NetworkUtils.SendPacket(networkConnection.GetClient().GetStream(),
                new Packet(idPacket, JsonConvert.SerializeObject(document.GetData(),
Formatting.Indented)));
        }
    }
}
using KVP_Api.Network.Packets;
using KVP_Api.Network.Utils;
using System.Net.Sockets;
using System.Text;
using System.Threading;

namespace KVP_Api.Network
{
    public class NetworkConnection
    {
        private ConnectableAddress connectableAddress;
        private TcpClient clientSocket;
        private Thread ctThread;
        private NetworkStream serverStream = default(NetworkStream);
        private PacketManager packetManager;


        public NetworkConnection(ConnectableAddress connectableAddress, PacketManager
packetManager)
        {
            this.connectableAddress = connectableAddress;
            clientSocket = new TcpClient();
            PacketManager = packetManager;        }

        public void Connect()
        {
            clientSocket.Connect(connectableAddress.GetHost(),
connectableAddress.GetPort());
            serverStream = clientSocket.GetStream();

            ctThread = new Thread(Recive);
            ctThread.Start();
        }

        private void Recive()
        {
            while (true)
            {
                if (NetworkUtils.IsConnected(clientSocket.Client))
                {
                    PacketManager.DispatchPacket(
                        NetworkUtils.RecivePacket(clientSocket.GetStream()), clientSocket);
                }
                else
                {
                    Thread.CurrentThread.Abort();
```

```
                    Disconect();
                }
            }
        }

        public void Disconect()
        {
            ctThread.Abort();
            clientSocket.Close();
        }

        public TcpClient GetClient()
        {
            return clientSocket;
        }

        public PacketManager PacketManager { get => packetManager; set => packetManager =
value; }
    }
}
namespace KVP_Api.Network
{
    public class ConnectableAddress
    {
        private string host;
        private int port;

        public ConnectableAddress(string host, int port)
        {
            this.host = host;
            this.port = port;
        }

        public string GetHost()
        {
            return host;
        }

        public int GetPort()
        {
            return port;
        }
    }
}
using KVP_Api.Network.Packets;
using System;
using System.IO;
using System.Net.Sockets;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

namespace KVP_Api.Network.Utils
{
    public static class NetworkUtils
    {
        public static bool IsConnected(this Socket socket)
        {
            try
            {
                return !(socket.Poll(1, SelectMode.SelectRead) && socket.Available == 0);
            }
            catch (SocketException) { return false; }
        }

        public static byte[] ObjectToByte(Object obj)
```

```
        {
            if (obj == null)
                return null;

            BinaryFormatter bf = new BinaryFormatter();
            MemoryStream ms = new MemoryStream();
            bf.Serialize(ms, obj);

            return ms.ToArray();
        }

        public static void SendPacket(NetworkStream strm, Packet packet)
        {
            IFormatter formatter = new BinaryFormatter();
            formatter.Serialize(strm, packet);
        }

        public static Packet RecivePacket(NetworkStream stream)
        {
            IFormatter formatter = new BinaryFormatter();
            return (Packet)formatter.Deserialize(stream);
        }
    }
}
using System;

namespace KVP_Api.Network.Packets
{
    [Serializable]
    public class Packet
    {

        private int id;

        private string data;

        public Packet(int id, string data)
        {
            this.id = id;
            this.data = data;
        }

        public int Id
        {
            get
            {
                return id;
            }
        }

        public string Data
        {
            get
            {
                return data;
            }
        }
    }
}
using KVP_Api.Network.Documents;
using System.Net.Sockets;

namespace KVP_Api.Network.Packets
{
    public abstract class PacketInHandler
```

```
    {
        protected int id;
        public abstract void InputHandler(Document data, TcpClient sender);
    }
}
using KVP_Api.Network.Documents;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Net.Sockets;

namespace KVP_Api.Network.Packets
{
    public class PacketManager
    {
        private Dictionary<int, PacketInHandler> packetInHandlers;

        public PacketManager()
        {
            packetInHandlers = new Dictionary<int, PacketInHandler>();
        }


        public void DispatchPacket(Packet packet, TcpClient client)
        {
            PacketInHandler handler = packetInHandlers[packet.Id];
            Document doc = new Document();
            doc.SetData(JsonConvert.DeserializeObject<Dictionary<string,
string>>(packet.Data));

            handler.InputHandler(doc, client);
        }


        public void RegisterHandler(int id, PacketInHandler packetInHandler)
        {
            if (!packetInHandlers.ContainsKey(id))
                packetInHandlers.Add(id, packetInHandler);
            //else
                //Console.WriteLine("Packet with id " + id + " all ready registered");
        }

        public void UnRegisterAllHandlers()
        {
            packetInHandlers.Clear();
        }

        public Dictionary<int, PacketInHandler> GetPackets() {
            return packetInHandlers;
        }
    }
}
namespace KVP_Api.Network.Packets
{
    public class PacketRC
    {
        public static int TEST = -100;
        public static int API = 100;
        public static int MESSAGE = 200;
        public static int PLAYER_DATA = 300;
        public static int GAME_DATA = 400;
    }
}
using System;
using System.Collections.Generic;
```

```
namespace KVP_Api.Network.Documents
{
    [Serializable]
    public class Document
    {
        private Dictionary<string, string> data;

        public Document()
        {
            data = new Dictionary<string, string>();
        }

        public void Add(string key, string value)
        {
            data.Add(key, value);
        }

        public void Remove(string key)
        {
            data.Remove(key);
        }

        public void SetData(Dictionary<string, string> data)
        {
            this.data = data;
        }

        public Dictionary<string, string> GetData()
        {
            return data;
        }
    }
}
```

Програма реалізація системи гравців

```
using KVP_Api.Player.Teams;
using Newtonsoft.Json;
using System.Net.Sockets;

namespace KVP_Api.Player
{
    public class Player
    {
        private int _id;
        private TcpClient _client;
        private PlayerInformation _playerInformation;


        public Player(int id, TcpClient client)
        {
            Id = id;
            Client = client;
        }

        public int Id { get => _id; set => _id = value; }
        [JsonIgnore]
        public TcpClient Client { get => _client; set => _client = value; }
        public PlayerInformation PlayerInformation { get => _playerInformation; set =>
_playerInformation = value; }
    }
}
using Newtonsoft.Json;
```

```
namespace KVP_Api.Player
{
    public class PlayerInformation
    {
        private int militaryUnit;
        private string name;
        private string surname;
        private string rank;
        private string position;

        public PlayerInformation(string name, string surname, int militaryUnit, string rank,
string position)
        {
            Name = name;
            Surname = surname;
            MilitaryUnit = militaryUnit;
            Rank = rank;
            Position = position;
        }

        public int MilitaryUnit { get => militaryUnit; set => militaryUnit = value; }
        public string Rank { get => rank; set => rank = value; }
        public string Position { get => position; set => position = value; }
        public string Name { get => name; set => name = value; }
        public string Surname { get => surname; set => surname = value; }
    }
}
```

Програмна реалізація системи команд

```
using System.Collections.Generic;

namespace KVP_Api.Player.Teams
{
    public class Team
    {
        private string name;
        private string displayName;
        private int size;

        private ISet<Player> players = new HashSet<Player>();

        public Team(string name, string displayName, int size)
        {
            Name = name;
            DisplayName = displayName;
            Size = size;
        }

        public string Name { get => name; set => name = value; }
        public string DisplayName { get => displayName; set => displayName = value; }
        public int Size { get => size; set => size = value; }
        public ISet<Player> Players { get => players; set => players = value; }
    }
}
```

```
using System.Collections.Generic;

namespace KVP_Api.Player.Teams
{
    public class TeamManager
    {
        private ISet<Team> teams;

        public TeamManager() => teams = new HashSet<Team>();
```

```
public ISet<Team> Teams { get => teams; set => teams = value; }

public TeamMarkers AddPlayer(Player player, string teamName)
{
    if (IsPlayerInTeam(player))
    {
        if (GetPlayerTeam(player).Name.Equals(teamName))
        {
            return TeamMarkers.PLAYER_ALREADY_IN_TEAM;
        }
        else
        {
            foreach (var t in Teams)
            {
                if (!t.Name.Equals(teamName))
                {
                    return TeamMarkers.TEAM_NOT_FOUND;
                }
                else if (t.Players.Count >= t.Size)
                {
                    return TeamMarkers.TEAM_IS_FULL;
                }

                RemovePlayer(player);
                t.Players.Add(player);
            }
        }
    }

    return TeamMarkers.GOOD;
}

public TeamMarkers SetPlayer(Player player, string teamName)
{
    foreach (var t in Teams)
    {
        if (!t.Name.Equals(teamName))
        {
            return TeamMarkers.TEAM_NOT_FOUND;
        }
        RemovePlayer(player);
        t.Players.Add(player);
    }

    return TeamMarkers.GOOD;
}

public void RemovePlayer(Player player)
{
    foreach (var t in Teams)
    {
        if (t.Players.Contains(player))
        {
            t.Players.Remove(player);
        }
    }
}

public Team GetPlayerTeam(Player player)
{
    foreach (Team t in Teams)
    {
        if (t.Players.Contains(player))
        {
            return t;
```

```
                }
            }

            return null;
        }

        public bool IsPlayerInTeam(Player player)
        {
            return GetPlayerTeam(player) != null ? true : false;
        }
    }
}

namespace KVP_Api.Player.Teams
{
    public enum TeamMarkers
    {
        PLAYER_ALREADY_IN_TEAM,
        TEAM_IS_FULL,
        TEAM_NOT_FOUND,
        GOOD
    }
}
```

Додаток В

Абстрактний клас для опису снаряду

```
using NewSDU.Artillery.Shell;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NewSDU
{
    public abstract class AbstractShell
    {
        private string name;
        private double q0;
        private double l;
        private double a;

        private List<ShellCharge> charges = new List<ShellCharge>();

        public AbstractShell(string name, double q0, double l, double a)
        {
            Name = name;
            Q0 = q0;
            L = l;
            A = a;
            create();
        }

        public abstract void create();

        public string Name { get => name; set => name = value; }
        public double Q0 { get => q0; set => q0 = value; }
        public double L { get => l; set => l = value; }
        public double A { get => a; set => a = value; }
        public List<ShellCharge> Charges { get => charges; set => charges = value; }
    }
}
```

Абстрактний клас для опису заряду снаряда

```
using NewSDU.Artillery.Shell.FormFact;

namespace NewSDU.Artillery.Shell
{
    public abstract class ShellCharge
    {
        private string _name;
        private int _V0;
        private double _Itz;
        private double _Iq;
        private double _rtetaQ;
        private double _rV0;
        private double _rC;
        private double _rw;

        private Formfactor formfactor;
```

```
        public ShellCharge(string name, int v0, double itz, double iq, double rtetaQ, double
rV0,
            double rC, double rw)
        {
            Name = name;
            V0 = v0;
            Itz = itz;
            Iq = iq;
            RtetaQ = rtetaQ;
            RV0 = rV0;
            RC = rC;
            Rw = rw;
            create();
        }

        public abstract void create();

        public string Name { get => _name; set => _name = value; }
        public int V0 { get => _V0; set => _V0 = value; }
        public double Itz { get => _Itz; set => _Itz = value; }
        public double Iq { get => _Iq; set => _Iq = value; }
        public double RtetaQ { get => _rtetaQ; set => _rtetaQ = value; }
        public double RV0 { get => _rV0; set => _rV0 = value; }
        public double RC { get => _rC; set => _rC = value; }
        public double Rw { get => _rw; set => _rw = value; }
        public Formfactor Formfactor { get => formfactor; set => formfactor = value; }
    }
}
```

Абстрактний клас для опису артилерійської установки

```
using System.Collections.Generic;

namespace NewSDU
{
    public abstract class AbstractArtillery
    {
        private string name;
        private double d;
        private double eta;

        private List<AbstractShell> shells = new List<AbstractShell>();

        public AbstractArtillery(string name, double d, double eta)
        {
            Name = name;
            D = d;
            Eta = eta;
            create();
        }

        public abstract void create();

        internal List<AbstractShell> Shells { get => shells; set => shells = value; }
        public double D { get => d; set => d = value; }
        public double Eta { get => eta; set => eta = value; }
        public string Name { get => name; set => name = value; }
    }
}
```

Клас для занесення параметрів снаряду

```
namespace NewSDU.Artillery.Shell.FormFact
```

```
{
    class FormfactorData
    {
        private double first;
        private double second;

        public FormfactorData(double first, double second)
        {
            First = first;
            Second = second;
        }

        public double First { get => first; set => first = value; }
        public double Second { get => second; set => second = value; }
    }
}

using System.Collections.Generic;

namespace NewSDU.Artillery.Shell.FormFact
{
    public class Formfactor
    {
        private Dictionary<double, FormfactorData> formfactors = new Dictionary<double,
FormfactorData>();

        public Dictionary<double, FormfactorData> Formfactors { get => formfactors; set =>
formfactors = value; }
    }
}
```

Приклад опису одного снаряду та зарядів до нього

```
using NewSDU.Artillery.Shell;
using NewSDU.Artillery.Shell.FormFact;

namespace NewSDU.ArtSystems.Sys_1.Shells.ShellCharges_1
{

    public class chr_1 : ShellCharge
    {
        public chr_1() : base("Первый", 602, 0.00158, 0.24, 0.25, 0.23, 0.72, 0.25)
        {

        }

        public override void create()
        {
            Formfactor frmf = new Formfactor();
            frmf.Formfactors.Add(0.0872665, new FormfactorData(0.9828, 0.0020));
            frmf.Formfactors.Add(0.261799, new FormfactorData(1.0174, 0.0020));
            frmf.Formfactors.Add(0.436332, new FormfactorData(1.0021, 0.0020));
            frmf.Formfactors.Add(0.785398, new FormfactorData(0.9865, 0.0014));
            frmf.Formfactors.Add(1.0472, new FormfactorData(0.9858, 0.0010));

            Formfactor = frmf;
        }
    }
}


using NewSDU.Artillery.Shell;
using NewSDU.Artillery.Shell.FormFact;
```

```
namespace NewSDU.ArtSystems.Sys_1.Shells.ShellCharges_1
{

    public class chr_2 : ShellCharge
    {
        public chr_2() : base("Второй", 509, 0.00029, 0.4, 0.25, 0.23, 0.72, 0.25)
        {

        }

        public override void create()
        {
            Formfactor frmf = new Formfactor();
            frmf.Formfactors.Add(0.0872665, new FormfactorData(0.9843, 0.0020));
            frmf.Formfactors.Add(0.261799, new FormfactorData(1.0507, 0.0020));
            frmf.Formfactors.Add(0.436332, new FormfactorData(1.0175, 0.0020));
            frmf.Formfactors.Add(0.785398, new FormfactorData(0.9882, 0.0014));
            frmf.Formfactors.Add(1.0472, new FormfactorData(0.9868, 0.0010));

            Formfactor = frmf;
        }
    }
}
using NewSDU.Artillery.Shell;
using NewSDU.Artillery.Shell.FormFact;

namespace NewSDU.ArtSystems.Sys_1.Shells.ShellCharges_1
{

    public class chr_3 : ShellCharge
    {
        public chr_3() : base("Третий", 423, 0.00029, 0.4, 0.25, 0.23, 0.79, 0.25)
        {

        }

        public override void create()
        {
            Formfactor frmf = new Formfactor();
            frmf.Formfactors.Add(0.0872665, new FormfactorData(1.0234, 0.0020));
            frmf.Formfactors.Add(0.261799, new FormfactorData(1.0920, 0.0020));
            frmf.Formfactors.Add(0.436332, new FormfactorData(1.0352, 0.0020));
            frmf.Formfactors.Add(0.785398, new FormfactorData(0.9895, 0.0014));
            frmf.Formfactors.Add(1.0472, new FormfactorData(0.9884, 0.0010));

            Formfactor = frmf;
        }
    }
}


using NewSDU.Artillery.Shell;
using NewSDU.Artillery.Shell.FormFact;

namespace NewSDU.ArtSystems.Sys_1.Shells.ShellCharges_1
{

    public class chr_4 : ShellCharge
    {
        public chr_4() : base("Четвертый", 381, 0.00029, 0.4, 0.25, 0.23, 0.81, 0.25)
        {

        }

        public override void create()
```

```
        {
            Formfactor frmf = new Formfactor();
            frmf.Formfactors.Add(0.0872665, new FormfactorData(1.0203, 0.0020));
            frmf.Formfactors.Add(0.261799, new FormfactorData(1.1109, 0.0020));
            frmf.Formfactors.Add(0.436332, new FormfactorData(1.0484, 0.0020));
            frmf.Formfactors.Add(0.785398, new FormfactorData(0.9805, 0.0014));
            frmf.Formfactors.Add(1.0472, new FormfactorData(0.9835, 0.0010));

            Formfactor = frmf;
        }
    }
}
using NewSDU.Artillery.Shell;
using NewSDU.Artillery.Shell.FormFact;

namespace NewSDU.ArtSystems.Sys_1.Shells.ShellCharges_1
{

    public class chr_5 : ShellCharge
    {
        public chr_5() : base("Пятый", 334, 0.00038, 0.4, 0.25, 0.23, 0.83, 0.25)
        {

        }

        public override void create()
        {
            Formfactor frmf = new Formfactor();
            frmf.Formfactors.Add(0.0872665, new FormfactorData(0.8578, 0.0020));
            frmf.Formfactors.Add(0.261799, new FormfactorData(1.0115, 0.0020));
            frmf.Formfactors.Add(0.436332, new FormfactorData(0.9657, 0.0020));
            frmf.Formfactors.Add(0.785398, new FormfactorData(0.9192, 0.0014));
            frmf.Formfactors.Add(1.0472, new FormfactorData(0.9120, 0.0010));

            Formfactor = frmf;
        }
    }
}
using NewSDU.Artillery.Shell;
using NewSDU.Artillery.Shell.FormFact;

namespace NewSDU.ArtSystems.Sys_1.Shells.ShellCharges_1
{

    public class chr_6 : ShellCharge
    {
        public chr_6() : base("Шестой", 282, 0.00039, 0.44, 0.25, 0.23, 0.83, 0.25)
        {

        }

        public override void create()
        {
            Formfactor frmf = new Formfactor();
            frmf.Formfactors.Add(0.0872665, new FormfactorData(0.7703, 0.0020));
            frmf.Formfactors.Add(0.261799, new FormfactorData(0.9289, 0.0020));
            frmf.Formfactors.Add(0.436332, new FormfactorData(0.9250, 0.0020));
            frmf.Formfactors.Add(0.785398, new FormfactorData(0.9172, 0.0014));
            frmf.Formfactors.Add(1.0472, new FormfactorData(0.9114, 0.0010));

            Formfactor = frmf;
        }
    }
}
using NewSDU.Artillery.Shell;
```

```
using NewSDU.Artillery.Shell.FormFact;

namespace NewSDU.ArtSystems.Sys_1.Shells.ShellCharges_1
{

    public class chr_full : ShellCharge
    {
        public chr_full() : base("Полный", 651, 0.00158, 0.24, 0.25, 0.23, 0.7, 0.25)
        {

        }

        public override void create()
        {
            Formfactor frmf = new Formfactor();
            frmf.Formfactors.Add(0.0872665, new FormfactorData(0.9375, 0.0020));
            frmf.Formfactors.Add(0.261799, new FormfactorData(0.9992, 0.0020));
            frmf.Formfactors.Add(0.436332, new FormfactorData(0.9931, 0.0020));
            frmf.Formfactors.Add(0.785398, new FormfactorData(0.9855, 0.0014));
            frmf.Formfactors.Add(1.0472, new FormfactorData(0.9839, 0.0010));

            Formfactor = frmf;
        }
    }
}
```

Приклад опису артилерійських систем

```
namespace NewSDU.ArtSystems.Sys_1.Shells
{
    class snarad_1 : AbstractShell
    {
        public snarad_1() : base("С6-1", 39.7, 0.709, 0.1190)
        {

        }

        public override void create()
        {

        }
    }
}
using NewSDU.ArtSystems.Sys_1.Shells.ShellCharges_1;

namespace NewSDU.ArtSystems.Sys_1.Shells
{
    class snarad_2 : AbstractShell
    {
        public snarad_2() : base("ОФ-540", 43.56, 0.709, 0.1190)
        {

        }

        public override void create()
        {
            Charges.Add(new chr_full());
            Charges.Add(new chr_1());
            Charges.Add(new chr_2());
            Charges.Add(new chr_3());
            Charges.Add(new chr_4());
            Charges.Add(new chr_5());
            Charges.Add(new chr_6());
        }
    }
```

```
}
using NewSDU.ArtSystems.Sys_1.Shells;

namespace NewSDU.ArtSystems
{
    class SG_2C3 : AbstractArtillery
    {
        public SG_2C3() : base("СГ 2С3", 0.152, 25)
        {

        }

        public override void create()
        {
            Shells.Add(new snarad_2());
            Shells.Add(new snarad_1());
        }
    }
}
```